Fixed-Parameter Approximation Schemes for Weighted Flowtime

Andreas Wiese[1]
Department of Industrial Engineering and Center for Mathematical Modeling,
Universidad de Chile, Chile
awiese@dii.uchile.cl

Abstract

Given a set of n jobs with integral release dates, processing times and weights, it is a natural and important scheduling problem to compute a schedule that minimizes the sum of the weighted flow times of the jobs. There are strong lower bounds for the possible approximation ratios. In the non-preemptive case, even on a single machine the best known result is a $O(\sqrt{n})$-approximation which is best possible. In the preemptive case on m identical machines there is a $O(\log \min\{\frac{n}{m}, P\})$-approximation (where P denotes the maximum job size) which is also best possible.

We study the problem in the parametrized setting where our parameter k is an upper bound on the maximum (integral) processing time and weight of a job, a standard parameter for scheduling problems. We present a $(1+\epsilon)$-approximation algorithm for the preemptive and the non-preemptive case of minimizing weighted flow time on m machines with a running time of $f(k, \epsilon, m) \cdot n^{O(1)}$, i.e., our combined parameters are k, ϵ, and m. Key to our results is to distinguish time intervals according to whether in the optimal solution the pending jobs have large or small total weight. Depending on this we employ dynamic programming, linear programming, greedy routines, or combinations of the latter to compute the schedule for each respective interval.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases Scheduling, fixed-parameter algorithms, approximation algorithms, approximation schemes

Digital Object Identifier 10.4230/LIPIcs.APPROX-RANDOM.2018.28

1 Introduction

A typical setting in scheduling is that one is given a set of n jobs J where each job $j \in J$ is characterized by a release date $r_j \in \mathbb{N}$, a processing time $p_j \in \mathbb{N}$, and a weight $w_j \in \mathbb{N}$. One seeks to compute a preemptive or non-preemptive schedule on m machines in which no job j is processed before its release date r_j and where each job can be processed by at most one machine at a time. Throughout this paper we will assume that the machines are identical. A natural objective function is to minimize the sum of the weighted flow times of the jobs: for C_j being the completion time of job j, the objective function is to minimize $\sum_j w_j(C_j - r_j)$.

Weighted flow time is a well-studied objective function. In the preemptive setting, the best known result on one machine is a $O(\log \log P)$-approximation algorithm [5] (where P denotes the maximum processing time of a job in the given instance) and for multiple machines in the unweighted case there is a $O(\log \min\{\frac{n}{m}, P\})$-approximation [15] and there is a lower bound of $\Omega(\log^{1-\epsilon} P)$ [11]. In the non-preemptive setting, even on one machine the best known approximation ratio is only $O(\sqrt{n})$ and this is best possible [13]. Hence, one can achieve only

[1] This work was partially supported by the Millennium Nucleus Information and Coordination in Networks ICM/FIC RC130003 and the grant Fondecyt Regular 1170223.

superconstant approximation ratios, apart from the preemptive case on one machine where in a recent break-through result a pseudo-polynomial time $O(1)$-approximation algorithm was found [6] and it is open to get a polynomial time $O(1)$-approximation (or even a PTAS) which is considered a long-standing important open problem.

Apart from approximation algorithms, another way to approach NP-hard problems is fixed-parameter tractability (FPT). One identifies a parameter k which intuitively measures the difficulty of a given instance and designs an exact algorithm with a running time of $f(k)n^{O(1)}$ for some computable function f. This is particularly appealing in settings like ours where there are strong lower bounds for the possible ratio of an approximation algorithm. Throughout this paper, we define $k := \max_j \max\{p_j, w_j\}$ which is a standard parameter in the literature for fixed-parameter algorithms for scheduling problems, see [18, 14]. Note that practical instances might have only a bounded range of job processing times and weights which makes this parameter relevant in these settings.

Recently, researchers started to combine the paradigms of approximation algorithms and fixed-parameter tractability (see, e.g., the survey by Marx [17]). Here, one constructs approximation algorithms that run in FPT time. For example, one constructs a $(1+\epsilon)$-approximation algorithm for instances with parameter k with a running time of $f(k,\epsilon) \cdot n^{O(1)}$ or $f(k) \cdot n^{O_\epsilon(1)}$, where $O_\epsilon(1)$ denotes a value that depends only on ϵ. This is in particular particularly interesting in cases where there can be no polynomial time $(1+\epsilon)$-approximation algorithm for arbitrary instances or where such an algorithm seems difficult to obtain. In this paper, we present such an algorithm for the weighted flow time problem where our combined parameter consists of the number of machines m, the quantity ϵ in the approximation ratio, and the value k as defined above.

1.1 Our contribution

We present $(1+\epsilon)$-approximation algorithms for the preemptive and the non-preemptive cases of minimizing weighted flow time on m machines with running times of $(mk)^{O(mk^3/\epsilon)} \cdot n^{O(1)}$ (preemptive case) and $(mk/\epsilon)^{O(mk^5)} \cdot n^{O(1)}$ (non-preemptive case), assuming that $p_j \leq k$ and $w_j \leq k$ for each job j. In particular, we obtain substantially better approximation factors than the known ratios of $O(\log\log P)$, $O(\log\min\{\frac{n}{m}, P\})$, and $O(\sqrt{n})$ for the respective settings.

To obtain our result in the preemptive case, the high level idea is the following: for each t denote by $W_{OPT}(t)$ the total weight of the pending jobs at time t in the optimal solution OPT, i.e., the total weight of the jobs that have been released but that have not yet been completed by time t. For time intervals during which $W_{OPT}(t)$ is small, in the optimal solution there can be only few pending jobs (bounded by a function of our parameters) and we can compute the optimal solution via a dynamic program. For the other intervals (during which $W_{OPT}(t)$ is large) we compute a preemptive schedule using a linear program. The LP is not exact since it uses a fractional objective function. However, we can guarantee that $W_{ALG}(t)$, the total weight of the pending jobs at time t in the computed solution ALG, is bounded by $W_{OPT}(t) + g(k,\epsilon,m)$ for some function g. Since $W_{OPT}(t)$ is large, the latter term is bounded by $(1+\epsilon)W_{OPT}(t)$. It is well-known that the objective function value of OPT equals $\int_t W_{OPT}(t)dt$. Therefore, we can argue that $ALG = \int_t W_{ALG}(t)dt \leq \int_t (1+\epsilon)W_{OPT}(t)dt = (1+\epsilon)OPT$. Since we do not know beforehand the values for t where $W_{OPT}(t)$ is large and small, we devise a dynamic program that scans the time axis from left to right step by step, guesses the schedules for the time intervals during which $W_{OPT}(t)$ is small, and computes the LP-solution for all intervals during which $W_{OPT}(t)$ is large. This yields a $(1+\epsilon)$-approximation for the preemptive setting.

In the non-preemptive case we again distinguish time intervals depending on whether $W_{OPT}(t)$ is large or small. For values of t where $W_{OPT}(t)$ is small, like before there are only few options for the pending jobs and, intuitively, we use a dynamic program to compute the optimal solution. For intervals I during which $W_{OPT}(t)$ is large we can no longer use the LP since it produces a schedule that is possibly preemptive. Even more, given I and a set of jobs J' that we want to complete within I (some of them might be released during I), it is not even clear how to determine whether there exists is a non-preemptive schedule for J' in I, even independent of the cost of such a schedule! To this end, we introduce the following strategy. If at a time t many jobs of the same *type* are pending, i.e., jobs with the same processing time and weight, then we schedule a large subset of them as soon as possible after time t on each machine. We say that they are scheduled as a *pack*. In case that there are several job types with many pending jobs, we give priority to the job type with the highest Smith ratio, i.e., with the largest ratio of weight divided by processing time. We show by an exchange argument that there indeed exists a feasible non-preemptive schedule that follows this rule at each time t (though we do not bound its cost at this point).

To bound our cost during an interval in which $W_{OPT}(t)$ is large, for the jobs scheduled as packs, we compare their cost with an optimal fractional schedule for them and prove that at each time t the total weight of the pending jobs in the two schedules differs by at most an additional term $\bar{g}(k, \epsilon, m)$ (for some function $\bar{g}$) which we can bound by $\epsilon W_{OPT}(t)$. For all other scheduled jobs we can show that each time t only few of them are pending and thus we can bound their total weight by another term $\epsilon W_{OPT}(t)$. This implies that $W_{ALG}(t) \leq (1+O(\epsilon))W_{OPT}(t)$ for each time t and hence our solution is $(1+\epsilon)$-approximative. We again devise a DP that scans the time axis from left to right and computes the solution ALG step by step, yielding a $(1+\epsilon)$-approximation algorithm for the non-preemptive case.

Note that in the non-preemptive case, already on one machine and with identical job weights the problem is NP-hard [13]. Hence, it is necessary to require that $p_j \leq k$ for each job j for the problem to be FPT in this case. On the other hand, no $W[1]$-hardness results are known for the settings of our FPT-algorithms above. Due to space constraints most proofs had to be omitted or moved to the appendix.

1.2 Other related work

For a single machine in the preemptive setting there is a QPTAS if the processing times and weights are in a quasi-polynomial range [8]. Its running time is $n^{O(\log W \log P/\epsilon^3)}$ (where W denotes the maximum job weight in the instance) and it is based on grouping the jobs according to processing times and weights. Note that the number of different groups appears in the exponent of n and hence we cannot use its methodology to obtain a running time of the form $f(k,\epsilon)n^{O(1)}$ or $f(k)n^{O_\epsilon(1)}$. The latter result was generalized by Bansal to an algorithm for a constant number of unrelated machines in the case when preemption is allowed by migration is forbidden, having a running time of $2^{(m \min\{\log nW, \log nP\}^3/\epsilon^3)}$ [3]. If $P \leq k$ and $W \leq k$ (like in our parametrized setting) this yields a running time of $2^{(m(\log nk)^3/\epsilon^3)}$ while our algorithms above for the preemptive case *with* migration and for the non-preemptive case (both being incomparable with Bansal's setting) have running times of $(mk)^{O(mk^3/\epsilon)} \cdot n^{O(1)} = 2^{O(\log(mk)\cdot mk^3/\epsilon + O(\log n))}$ and $(mk/\epsilon)^{O(mk^5)} \cdot n^{O(1)} = 2^{O(\log(mk/\epsilon)\cdot mk^5 + O(\log n))}$, respectively.

In the non-preemptive setting, on m machines one can achieve a ratio of $O(\sqrt{n/m}\log(n/m))$ and there is a lower bound of $\Omega(n^{1/3-\epsilon})$ [15]. Minimizing flow time was also studied in the online setting. In the unweighted case on one machine the SRPT algorithm is optimal and on parallel machines it achieves a competitive ratio of $O(\log \min\{\frac{n}{m}, P\})$ which is best possible [15, 11]. In the weighted case, for one machine there is an $O(\log^2 P)$-

competitive semi-online algorithm known [9] and for multiple machines there is a lower bound of $\Omega(\min\{\sqrt{P}, \sqrt{W}, (n/m)^{1/4}\})$ [9]. This was improved recently to a $O(\log(\min\{P, D, W\}))$-competitive algorithm [2] (where D is the maximum ratio of the job densities), using the $O(\log D)$-competitive algorithm from [4]. Many results are known in the resource augmentation setting in which the online algorithm is given faster machines than the adversary, see [16, Chapter 15] and references therein.

If all jobs are released at time zero, the problem is equivalent to minimizing the weighted sum of completion times. This problem admits a polynomial time $(1+\epsilon)$-approximation algorithm and in the preemptive case even an EPTAS [1]. This was generalized to PTASs for related machines [7] for which in the non-preemptive case there is also an EPTAS [10].

Approximation schemes with a running time of $f(k) \cdot n^{O_\epsilon(1)}$ (where the parameter k denotes the size of the desired solution) are known for Maximum Independent Set of Rectangles, Two-dimensional Geometric Knapsack with rotations [12], and for Unsplittable Flow on a Path [19] while PTASs are open for these problems.

2 Preemptive case

We present our $(1+\epsilon)$-approximation algorithm for minimizing the weighted flow time on m identical machines with preemption, i.e., $P|r_j, pmtn|\sum w_j F_j$. Our algorithm has a running time of $k^{O(m \cdot k^3/\epsilon)} n^{O(1)}$, assuming that $p_j, w_j \in \{1, ..., k\}$ for each job j, i.e., our combined parameter is $k + m + 1/\epsilon$. We proceed in two steps: first, we show that there is a structured solution OPT' such that $c(OPT') \leq (1+\epsilon)c(OPT)$. Then, we devise a dynamic program that computes a solution with this structure whose cost is at most the cost of OPT'.

We group the jobs into groups. For each $\ell, \ell' \in \{1, ..., k\}$ we define $J_{\ell,\ell'} := \{j | p_j = \ell \wedge w_j = \ell'\}$. First, we prove by some simple exchange arguments that for each job group the optimal solution OPT essentially processes its jobs in the order of their release dates and hence the number of partially processed jobs of each group is small. Throughout this paper, whenever we speak about the order of the release dates of the jobs, we break ties in a fixed arbitrary order. Also, we will assume that $1/\epsilon \in \mathbb{N}$ and define $\Gamma := \cup_{t \in \mathbb{N}_0} \epsilon t$.

▶ **Lemma 1.** *By losing a factor of $1+\epsilon$ in the approximation ratio, we can assume the following properties for OPT: for each $t \in \Gamma$, each machine i works on exactly one job during the entire interval $[t, t+\epsilon)$ or is idle during $[t, t+\epsilon)$. Also, for each job class $J_{\ell,\ell'}$ and each time t, the jobs of $J_{\ell,\ell'}$ that are partially but not completely scheduled at time t are among the mk/ϵ jobs with the earliest release dates pending at time t.*

For ease of notation, in the sequel we will denote by OPT the $(1+\epsilon)$-approximative solution due to Lemma 1.

2.1 Near-optimal solution

We define a near-optimal structured solution OPT'. For a schedule S and any $t \geq 0$ let $W_S(t)$ denote the total weight of the jobs j that satisfy $r_j \leq t$ but that are not finished by time t in S. We define $I_{\text{low}} := \bigcup_{t \in \Gamma : W_{OPT}(t) \leq 3mk^3/\epsilon^2} [t, t+\epsilon)$ and $I_{\text{high}} := \bigcup_{t \in \Gamma : W_{OPT}(t) > 3mk^3/\epsilon^2} [t, t+\epsilon)$. For each time interval $[t, t+\epsilon) \subseteq I_{\text{low}}$ with $t \in \Gamma$ we define the schedule OPT' to be identical to OPT. For the timesteps in I_{high} we consider each maximally large interval in I_{high}. Let $[t_1, t_2) \subseteq I_{\text{high}}$ be such an interval with $t_1, t_2 \in \Gamma$. In order to define OPT' during $[t_1, t_2)$, we compute a schedule using a linear program. For each job j, denote by $p'_j \in \mathbb{N}$ the amount of work of j that OPT finishes during $[t_1, t_2)$. Note that p'_j is an integral multiple of ϵ. Our linear program computes a schedule for $[t_1, t_2)$ such that we finish exactly p'_j units of work

of each job j during $[t_1, t_2)$, at each time $t \in [t_1, t_2)$ each job j is processed by at most one machine, and the objective function is a fractional objective in which we assume that we split each job j into p'_j/ϵ unit size pieces of weight $\epsilon w_j/p'_j$ each. This yields the following linear program that we denote by (LP).

$$\begin{aligned}
\min \quad & \sum_{j \in J} \sum_{t \in \{t_1, t_1+\epsilon, \dots, t_2-\epsilon\}} x_{ijt} \cdot (t+\epsilon)\epsilon \frac{w_j}{p'_j} \\
\text{s.t.} \quad & \sum_{i=1}^{m} \sum_{t \in \{t_1, t_1+\epsilon \dots, t_2-\epsilon\}} x_{ijt} = p'_j \quad \forall j \in J \\
& \sum_{i=1}^{m} x_{ijt} \leq 1 \quad \forall j \in J \, \forall t \in \{t_1, t_1+\epsilon, \dots, t_2-\epsilon\} \\
& \sum_{j \in J} x_{ijt} \leq 1 \quad \forall i \in [m] \, \forall t \in \{t_1, t_1+\epsilon, \dots, t_2-\epsilon\} \\
& x_{ijt} \geq 0 \quad \forall j \in J \, \forall i \in [m] \, \forall t \in \{r_j, r_j+\epsilon, \dots, t_2-\epsilon\}
\end{aligned}$$

We compute an optimal extreme point solution to (LP) in polynomial time. By interpreting (LP) as a model for a suitable instance of bipartite matching one can easily argue that this solution is integral. We simplify it to ensure that at each time at most mk/ϵ jobs from the same group are partially processed (where "partially" is defined with respect to the processing time p'_j), using some exchange arguments.

▶ **Lemma 2.** *Given an optimal integral solution to (LP), in time polynomial in n and k we can compute an optimal integral solution that has the property that at each time $t \in \{t_1, t_1+\epsilon, \dots, t_2-\epsilon\}$ from each job class $J_{\ell,\ell'}$ there are at most mk/ϵ jobs j such that $0 < \bar{p}_j(t) < p'_j$ where $\bar{p}_j(t)$ denotes the amount of j that has been processed during $[t_1, t)$.*

The schedule of OPT' during $[t_1, t_2)$ is now defined by applying Lemma 2 to the optimal integral solution to (LP). For x being the resulting solution, during an interval $[t, t+\epsilon) \subseteq [t_1, t_2)$ with $t \in \Gamma$ we schedule a job j on a machine i if and only if $x_{ijt} = 1$. We do the above procedure for each maximally large interval $[t_1, t_2) \subseteq I_{\text{high}}$ with $t_1, t_2 \in \Gamma$. This completes the definition of OPT'.

In order to show that OPT' has small cost compared to OPT, we use the following well-known way to measure the flow time objective function. We define the cost of a schedule S to be $c(S) := \sum_{j \in J} w_j(C_j^S - r_j)$ where C_j^S denotes the completion time of a job j in S.

▶ **Proposition 3.** *For any schedule S we have that $c(S) = \int_t W_S(t)dt$.*

By construction have that $W_{OPT'}(t) = W_{OPT}(t)$ for each $t \in I_{\text{low}}$. In each maximally large interval $[t_1, t_2) \subseteq I_{\text{high}}$ the solution OPT' equals the solution to (LP) which is the optimal solution for the fractional objective function. The latter differs from the integral objective function since there can be jobs that are partially but not completely processed. However, due to Lemmas 1 and 2 there can be at most $3mk/\epsilon$ such jobs from each class $J_{\ell,\ell'}$ and thus their total weight is bounded by $3mk^3/\epsilon$. However, $W_{OPT}(t) > 3mk^3/\epsilon^2$ if $t \in I_{\text{high}}$ and hence the total contribution of these jobs to $W_{OPT'}(t)$ is bounded by $\epsilon W_{OPT}(t)$ for each $t \in \Gamma$. Using Proposition 3 and additionally that the fractional cost of the LP-schedule is at most the cost of OPT in the same respective interval $[t_1, t_2)$, we can prove the following lemma.

▶ **Lemma 4.** *It holds that $c(OPT') \leq (1+\epsilon)c(OPT)$.*

2.2 Dynamic program

We give a dynamic program that computes a schedule whose cost is at most $c(OPT')$. It scans the time axis from left to right. Given a timestep $t \in \Gamma$ with $t \in I_{\text{low}}$ it intuitively guesses the schedule of OPT during $[t, t+\epsilon)$. If $t+\epsilon \in I_{\text{low}}$ then it proceeds with the timestep $t+\epsilon$. If $t+\epsilon \in I_{\text{high}}$ then it guesses the earliest timestep $t' \in \Gamma$ such that $t < t'$ and $t' \in I_{\text{low}}$, and additionally it guesses the characteristics of the pending jobs at time t'. For the time interval $[t, t')$ it computes a schedule using (LP). At each time $t \in I_{\text{low}}$ there are at most $3mk^3/\epsilon^2$ pending jobs in OPT and due to Lemma 1 there are only $f(k, m, \epsilon)$ many possibilities for their characteristics, for some function f. Hence, we can embed the whole procedure into a dynamic program that has $f(k, \epsilon, m)$ cells for each time $t \in \Gamma$.

We assume w.l.o.g. that $\min_j r_j = 1$ and that $\max_j r_j = O(n^2 k)$ since we can shift the release dates uniformly and if $\max_j r_j$ is too large then we can split the instance into independent subinstances. Note that then OPT' finishes its last job by time $T := \max_j r_j + nk$ the latest.

We define now our dynamic program formally. We say that a *configuration for a time* $t \in \Gamma$ specifies for each job its remaining processing time at time t. Formally, a configuration is a vector $(\bar{p}_j)_{j \in J}$ such that $0 \leq \bar{p}_j \leq p_j$ and $\bar{p}_j/\epsilon \in \mathbb{N}$ for each $j \in J$. For each time $t \in \Gamma$ there is a configuration that corresponds to the pending jobs of OPT at time t. We will show now that if $t \in I_{\text{low}}$ then there are only $mk^{O(mk^3/\epsilon)}$ possibilities for this configuration, using that there can be only $3mk^3/\epsilon^2$ pending jobs at time t.

▶ **Lemma 5.** *For each time $t \in \Gamma$ we can compute in time $(mk)^{O(mk^3/\epsilon)} n^{O(1)}$ a set $\mathcal{C}(t)$ of at most $(mk)^{O(mk^3/\epsilon)}$ configurations such that if $t \in I_{\text{low}} \cap \Gamma$ then one of them corresponds to OPT at time t. The set $\mathcal{C}(T)$ contains only the configuration c_T in which all jobs have completed.*

Proof. If $t \in I_{\text{low}} \cap \Gamma$ then $W_{OPT}(t) \leq 3mk^3/\epsilon^2$ and hence there can be at most $3mk^3/\epsilon^2$ jobs pending at time t. For each of them there are only k^2 options which class it belongs to, k/ϵ possibilities for how much processing time is left, and then by Lemma 1 only $3mk^3/\epsilon^2 + mk/\epsilon$ possibilities for its identiy. This yields $(k^3/\epsilon(3mk^3/\epsilon^2 + mk/\epsilon))^{3mk^3/\epsilon^2} = (mk)^{O(mk^3/\epsilon)}$ combinations. For each of them we can compute the corresponding set of jobs in time $n^{O(1)}$, yielding a running time of $(mk)^{O(mk^3/\epsilon)} n^{O(1)}$ for the computation of all combinations. ◀

Our dynamic program has a cell (t, c) for each combination of a time $t \in \Gamma \cap [0, ..., T]$ and a configuration $c \in \mathcal{C}(t)$. We say that a job is *pending in* c if it has not yet been completed according to c. The entry of the cell (t, c) stores a schedule for the time interval $[t, T)$ in which the remaining parts of the pending jobs in c are scheduled and additionally all jobs j with $r_j > t$. Then the cell $(0, c_0)$ stores a schedule for the whole instance where c_0 denotes the configuration in which no job has been worked on yet, i.e., $\bar{p}_j = p_j$ for each job j.

For the base case, we assign to the cell (T, c_T) the empty schedule. Suppose we are given a cell (t, c) with $t < T$. We compute its schedule as follows. The reader may imagine that $t \in I_{\text{low}}$ and that c is the configuration of OPT' (and hence OPT) at time t. Intuitively, we guess the schedule of OPT' during $[t, t+\epsilon)$. Formally, we try all possibilities for up to m jobs $j_1, ..., j_{m'}$ that are pending in c and that are among the mk/ϵ jobs with earliest release dates among the pending jobs at time t of their respective job class (note that we can assume that OPT' does not schedule other jobs during $[t, t+\epsilon)$, see Lemma 1). In particular, the number of schedules to enumerate is bounded by $g(k, \epsilon, m)$ for some function g. Each guess of a set of jobs $j_1, ..., j_{m'}$ to be processed during $[t, t+\epsilon)$ yields a configuration c' for time $t+\epsilon$. If $c' \in \mathcal{C}(t+\epsilon)$ then we append the schedule stored in $DP(t+\epsilon, c')$ to the schedule

for $[t, t+\epsilon)$. Note that in case that OPT' is in configuration c at time t and we guessed $j_1, ..., j_{m'}$ correctly, for the interval $[t, t+\epsilon)$ we obtain the same schedule as OPT' (up to permutation of machines) and at time $t+\epsilon$ the schedule OPT' is in configuration c'.

If $c' \notin \mathcal{C}(t+\epsilon)$ then we guess the smallest time $t'' \in \Gamma$ with $t'' > t$ such that $t'' \in I_{\text{low}}$ and additionally the configuration c'' of OPT' at time t''. For the time interval $[t+\epsilon, t'')$ we compute a schedule using (LP): for each job $j \in J$ denote by $\bar{p}'_j$ its remaining processing time at time $t+\epsilon$ according to c' and by $\bar{p}''_j$ its remaining processing time at time t'' according to c''. We solve (LP) where for each job j we define $p'_j := \bar{p}'_j - \bar{p}''_j$. If the LP does not have a solution then we reject our guess. Otherwise, we apply Lemma 2 to the resulting solution. We append the schedule in $DP(t'', c'')$ to the computed schedules for $[t, t+\epsilon)$ and for $[t+\epsilon, t'')$. Note that if all guesses were correct, then for $[t, t'')$ we obtain the same schedule as OPT' (up to permutation of machines). Each enumerated guess yields a schedule for the jobs pending at time t in c (completing their remaining processing time according to c) and the jobs j with $r_j > t$ (processing them completely). Among all these solutions, we store in $DP(t, c)$ the solution that minimizes the sum of the weighted flow times of all the latter jobs.

One can show by induction over the DP-cells that the cost of the solution in the cell $(0, c_0)$ (note that $c_0 \in \mathcal{C}(0)$ since $\min_j r_j = 1$) is bounded by $c(OPT')$: whenever we process a DP-cell (t, c) such that OPT' is in configuration c at time t, among our enumerated guesses are the jobs that OPT processes during $[t, t+\epsilon)$, and t'' and c'' according to OPT' which then yields the same schedule for $[t, t'')$ as OPT'. The number of DP-cells is bounded by the total number of configurations for all relevant time steps, see Lemma 5. Moreover, we can compute the entry for each DP-cell by performing $(mk)^{O(mk^3/\epsilon)} \cdot n^{O(1)}$ guesses (for the jobs jobs $j_1, ..., j_{m'}$, t'' and c'') and a computation of $n^{O(1)}$ for each guess. This yields the following theorem.

▶ **Theorem 6.** *For the problem of minimizing the weighted flow time on m identical machines with preemption and job release dates, $P|r_j, pmtn|\sum w_j F_j$, there is a $(1+\epsilon)$-approximation algorithm with a running time of $(mk)^{O(mk^3/\epsilon)} \cdot n^{O(1)}$, assuming that $p_j, w_j \in \{1, ..., k\}$ for each job j.*

3 Non-preemptive case

In this section we present a $(1+\epsilon)$-approximation algorithm for the non-preemptive setting, i.e, for $P|r_j|\sum w_j F_j$, whose running time is bounded by $f(k, \epsilon, m) \cdot n^{O(1)}$ for some function f, assuming that $p_j, w_j \in \{1, ..., k\}$ for each job j.

Similar to Lemma 1 in the preemptive case, we can assume that OPT schedules the jobs in order of their release dates.

▶ **Lemma 7.** *We can assume that if OPT starts a job $j \in J_{\ell,\ell'}$ before a job $j' \in J_{\ell,\ell'}$ then r_j is released before $r_{j'}$, breaking ties in an arbitrary fixed order. Also, each job starts and ends at an integral time point.*

3.1 Near-optimal solution

Like in the preemptive case, we first define a structured solution OPT' with $c(OPT') \leq (1+\epsilon)c(OPT)$ and then provide a dynamic program that finds a solution with cost at most $c(OPT')$. To define OPT', we split the time horizon into two parts I'_{low} and I'_{high}. We define $I'_{\text{low}} := \bigcup_{t \in \mathbb{N}: W_{OPT}(t) \leq 2^{2k^2} m^2/\epsilon} [t, t+1)$ and $I'_{\text{high}} := \bigcup_{t \in \mathbb{N}: W_{OPT}(t) > 2^{2k^2} m^2/\epsilon} [t, t+1)$. For each time step $[t, t+1) \subseteq I'_{\text{low}}$ with $t \in \mathbb{N}$ we define OPT' to be identical to OPT. Let $[t_1, t_2) \subseteq I'_{\text{high}}$ be a maximally large interval of I'_{high} (and thus $t_1, t_2 \in \mathbb{N}$). Our goal is now

to define a structured schedule for $[t_1, t_2)$ whose cost is by at most a factor $1 + \epsilon$ larger than the cost of OPT during $[t_1, t_2)$. Let J' denote the jobs that in OPT are started and completed during $[t_1, t_2)$. In the preemptive case we defined a schedule for J' during $[t_1, t_2)$ via (LP). However, an LP-solution might preempt (and migrate) jobs. Instead, we look for a structured non-preemptive solution that our dynamic program can easily compute later. Note that if we are given the interval $[t_1, t_2)$ with the set of jobs J' it is not even clear how compute *any* non-preemptive schedule, even independent of the cost.

We define a structured schedule that has the *pack-property*. Intuitively, this property implies that if at some time t there are many pending jobs of the same class then we schedule a *pack*, i.e., a large subset of them, as soon as we can. Formally, we say that a job j is *ready* at time t if $r_j \leq t$ and if j has not been started before t. A set of ready jobs $Q \subseteq J'$ forms a *pack* if all jobs in Q are of the same class and if $p(Q) = m \cdot k!$, where for any set of jobs $\tilde{J}$ we define $p(\tilde{J}) := \sum_{j \in \tilde{J}} p_j$.

▶ **Definition 8** (Pack-property)**.** A schedule for the jobs J' in the interval $[t_1, t_2)$ has the *pack-property* if for each integral time $t \in [t_1, t_2)$ at which a machine finishes a job or such that there is machine that is idle during $[t-1, t)$ the following holds:

- If for some job class $J_{\ell,\ell'}$ there are jobs from $J' \cap J_{\ell,\ell'}$ with a total processing time of at least $4m^2kk!$ that are ready at time t, we say that the class $J_{\ell,\ell'}$ is *waiting*. For a waiting job class $J_{\ell,\ell'}$ let $Q_{\ell,\ell'}$ denote the $m \cdot k!/\ell$ ready jobs with earliest release dates among the ready jobs in $J_{\ell,\ell'}$.
- Among the waiting jobs classes, let $J_{\ell,\ell'}$ be a waiting job class with highest density, i.e., that maximizes ℓ'/ℓ, breaking ties in a fixed arbitrary way. For each machine i let $t(i)$ denote the earliest time $t' \geq t$ when i is not processing a job that it is executing during $[t-1, t)$. We require that then each machine i schedules $k!/\ell$ jobs from $Q_{\ell,\ell'}$ during $[t(i), t(i) + k!)$. In the resulting schedule, we call the jobs in $Q_{\ell,\ell'}$ *pack-jobs* and we say that they are *scheduled as the pack* $Q_{\ell,\ell'}$.

We first prove that a schedule for $[t_1, t_2)$ with the pack-property always exists and afterwards we will bound its cost. We will do this via a repeated exchange argument: starting with the optimal schedule, if at some time t the schedule does not obey the pack-property then we move some jobs in the schedule such that the pack-property holds at time t. Note that in OPT a machine might work on a job $j \notin J'$ during $[t_1, t_2)$ if j is started before t_1 or completed after t_2. For each machine i, denote by $t_1(i)$ and $t_2(i)$ the earliest and latest times in $[t_1, t_2]$ at which i does not work on a job in J'.

▶ **Lemma 9.** *There is a schedule for the jobs J' during the interval $[t_1, t_2)$ that fulfills the pack-property such that each machine i works on jobs in J' only during $[t_1(i), t_2(i))$.*

We apply Lemma 9 to each maximally large interval $[t_1, t_2) \subseteq I'_{\text{high}}$ and its corresponding job set J'. Then we possibly swap jobs from the same class such that the resulting schedule satisfies Lemma 7. Denote by OPT' the resulting schedule.

3.2 Bounding the cost

We want to prove that $c(OPT') \leq (1 + \epsilon)c(OPT)$. The intuition is the following: recall that in a schedule with the pack-property some job are pack-jobs. The *non-pack-jobs* are all other jobs that are scheduled during a maximally large interval $[t_1, t_2) \subseteq I'_{\text{high}}$, denoted by J_{np}. At each time $t \in I'_{\text{high}}$ there can be only few pending jobs in J_{np} since if the total processing time of pending non-pack jobs from one class was at least $4m^2kk!$ then some of them would be pack-jobs. Hence, the contribution to $W_{OPT'}(t)$ of the non-pack-jobs is negligible for each $t \in I'_{\text{high}}$ and hence the same holds for their contribution to the objective function.

▶ **Proposition 10.** *For each time $t \in I'_{\text{high}}$ let $W^{\text{np}}_{OPT'}(t)$ denote the total weight of the pending jobs in J_{np} at time t in OPT' and then it holds that $W^{\text{np}}_{OPT'}(t) \leq 4m^2k^3k! \leq \epsilon \cdot W_{OPT}(t)$. Therefore, $\int_{t \in I'_{\text{high}}} W^{\text{np}}_{OPT'}(t)dt \leq \epsilon \cdot OPT$.*

Let J_{p} denote the jobs that are scheduled as pack-jobs during a maximally large interval $[t_1, t_2) \subseteq I'_{\text{high}}$. When in OPT' a set of jobs $\bar{Q}_{\ell,\ell'}$ is scheduled as a pack then for each machine i we can identify a time $t(i)$ such that the jobs from $\bar{Q}_{\ell,\ell'}$ are scheduled exactly during $[t(i), t(i) + k!)$. It can happen that $t(i) \neq t(i')$ for two machines i, i'. In order to simplify the analysis, we define a new schedule OPT''_{p} only for the pack-jobs in which the latter does *not* happen and such that the cost of OPT''_{p} is similar to the cost of the pack jobs in OPT'. Intuitively, OPT''_{p} is a greedy schedule: one can interpret the jobs from each pack as m super-jobs with processing time $k!$ each such that each machine is assigned one of these super-jobs. In our schedule, these super-jobs are scheduled greedily by their density (or, equivalently, by their weight).

Let $Q_1, ..., Q_s$ be a partition of the pack-jobs such that for each r the jobs in Q_r are scheduled as the pack Q_r. For each pack Q_r denote by t_r the earliest start time of a job in Q_r in OPT'. In OPT''_{p} we define that on each machine i for each pack Q_r the jobs in Q_r are scheduled during $[t_r, t_r + k!)$. Intuitively, to achieve this we shift some of the jobs of Q_r in OPT' to the left. Denote by OPT''_{p} the resulting schedule. At time t_r all jobs from Q_r are already released and thus in OPT''_{p} no job is scheduled before its release date. Together with the following lemma this implies that OPT''_{p} is feasible.

▶ **Lemma 11.** *For any two packs $Q_r, Q_{r'}$ with $r \neq r'$ we have that $[t_r, t_r + k!) \cap [t_{r'}, t_{r'} + k!) = \emptyset$.*

Proof. Assume w.l.o.g. that $t_r \leq t_{r'}$. In OPT', for the pack Q_r there is a machine i such that the jobs from Q_r are scheduled during $[t_r, t_r + k!)$ and for any machine $i' \neq i$ the jobs from Q_r are scheduled during $[t_r + \alpha_{i'}, t_r + \alpha_{i'} + k!)$ for some $\alpha_{i'} \in \{0, ..., k-1\}$. Hence, $t_{r'} \geq t_r + k!$. ◀

When we constructed OPT''_{p} based on OPT', we shifted each job by at most $k - 1$ units to the left and hence the cost of the schedule changes only marginally.

▶ **Lemma 12.** *We have that $W^{\text{p}}_{OPT'}(t) \leq W_{OPT''_{\text{p}}}(t) + k^2m \leq W_{OPT''_{\text{p}}}(t) + \epsilon W_{OPT}(t)$ where $W^{\text{p}}_{OPT'}(t)$ denotes the total weight of the pending pack-jobs in OPT' at time t.*

Proof. Observe that in OPT' and OPT''_{p} the completion time of a job differs by at most k. Hence, at each point in time OPT''_{p} has completed at most km jobs more than OPT' and the total weight of these jobs is at most k^2m. Therefore, $W^{\text{p}}_{OPT'}(t) \leq W_{OPT''_{\text{p}}}(t) + k^2m \leq W_{OPT''_{\text{p}}}(t) + \epsilon W_{OPT}(t)$ for each $t \in I'_{\text{high}}$. ◀

We want to prove now that OPT''_{p} has small cost. Consider a maximally large interval $[t_1, t_2) \subseteq I'_{\text{high}}$. We define an artificial instance I_{p} consisting of only the pack-jobs scheduled during $[t_1, t_2)$. We assign an artificial release date t'_r to all jobs in each pack Q_r. Intuitively, t'_r is defined to be the earliest time when we might schedule the jobs in Q_r as pack-jobs because at this time all jobs in Q_r and many other jobs of the same class with later release dates have already been released. Formally, assume that $J_{\ell,\ell'}$ is the (unique) job class such that Q_r contains jobs from $J_{\ell,\ell'}$. We define t'_r to be the earliest time when all jobs in Q_r are released and the total processing time of already released jobs $j \in J_{\ell,\ell'}$ with $r_j \geq \min_{j' \in Q_r} r_j$ is at least $4m^2kk!$. Note that OPT' does not process a job in Q_r earlier than t'_r since in OPT' the jobs in Q_r are processed as pack jobs. We consider the solution to (LP) for this instance where we define $p'_j := p_j$ for each $j \in J_{\text{p}}$. Let x^* be this solution and let $FRAC$

be the resulting fractional schedule. We partition the jobs J_p into groups $J_\mathrm{p}^{(1)}, J_\mathrm{p}^{(2)}, ..., J_\mathrm{p}^{(\gamma)}$ according to their densities: for each group $J_\mathrm{p}^{(g)}$ all jobs $j \in J_\mathrm{p}^{(g)}$ have exactly the same density w_j/p_j and for two jobs j, j' with $j \in J_\mathrm{p}^{(g)}$ and $j' \in J_\mathrm{p}^{(g')}$ with $g < g'$ we have that $w_j/p_j < w_{j'}/p_{j'}$. In particular, jobs in different groups have different densities.

▶ **Lemma 13.** *We can assume w.l.o.g. that in $FRAC$ for each $t \in I'_\mathrm{high}$ with $t \in \mathbb{N}$ we have that during $[t, t+1)$ either all machines are idle or all machines work on jobs from a pack Q_r such that each job $j \in Q_r$ has the highest density w_j/p_j among all pending jobs at time t. Moreover, for each group $J_\mathrm{p}^{(g)}$ at each time t there is at most one pack Q_r containing jobs from $J_\mathrm{p}^{(g)}$ from which some jobs or some parts of some jobs have already been processed, but not all jobs completely.*

Proof. The number of jobs in each pack is $mk!/\ell$ where ℓ denotes the size of each job in the pack and, in particular, this value is divisible by m. Also, a solution to (LP) is optimal if for each $t \in I'_\mathrm{high}$ during $[t, t+1)$ each machine works on a job j with the largest density w_j/p_j among all available jobs. Hence, using a greedy algorithm we can construct an optimal fractional solution with the latter property that satisfies the claim of the lemma. ◀

For each $t \in I'_\mathrm{high}$ denote by $\tilde{W}_{FRAC}(t)$ the total fractional pending weight of the jobs j with $r_j \leq t$ that have not completed by time t in $FRAC$. Formally, for each job $j \in J_\mathrm{p}$ and each $t \in I'_\mathrm{high}$ let $p_j(t)$ denote the remaining processing time of job j at time t in $FRAC$. We define $\tilde{W}_{FRAC}(t) := \sum_{j \in J_\mathrm{p}} p_j(t) \frac{w_j}{p_j}$.

In the next lemma, we prove that $\tilde{W}_{FRAC}(t)$ is not too large compared to $W_{OPT}(t)$ for each t. Note that even though $\tilde{W}_{FRAC}(t)$ denotes the fractional remaining weight which is not larger than the integral remaining weight, for some t it can be that $W_{OPT}(t) < \tilde{W}_{FRAC}(t)$ because OPT can work on a job j in a pack Q_r before $FRAC$ can do this since possibly $r_j < t'_r$. However, for each job class $J_{\ell,\ell'}$ the total weight of such jobs is bounded by $4m^2k^3k!$ at each time t which we will use to prove the following lemma.

▶ **Lemma 14.** *For each group $J_\mathrm{p}^{(g)}$ and each time $t \in \{t_1, ..., t_2 - 1\}$ we have that $\tilde{W}_{FRAC}(t) \leq W_{OPT}(t) + 4m^2k^5k!$.*

In the next lemma, we show that $W_{OPT'_p}(t)$ is not much larger than $\tilde{W}_{FRAC}(t)$ which we will eventually use in order to bound the cost of OPT'_p and hence of OPT'. For a set of jobs $\bar{J}$ denote by $W^{\bar{J}}_{OPT''_\mathrm{p}}(t)$ and $\tilde{W}^{\bar{J}}_{FRAC}(t)$ the integral and fractional weight of its pending jobs at time t in OPT'_p and $FRAC$, respectively.

▶ **Lemma 15.** *For each group $J_\mathrm{p}^{(g)}$ and each $t \in \{t_1, ..., t_2 - 1\}$ we have that $W^{J_\mathrm{p}^{(g)}}_{OPT''_\mathrm{p}}(t) \leq \tilde{W}^{J_\mathrm{p}^{(g)}}_{FRAC}(t) + 2^{\gamma-g}mk \cdot k!$. Therefore, $W_{OPT''_\mathrm{p}}(t) \leq \tilde{W}_{FRAC}(t) + 2^{k^2}mk \cdot k! \leq W_{OPT''_\mathrm{p}}(t) + \epsilon W_{OPT}(t)$.*

Using Lemmas 14 and 15 we bound the cost of OPT'.

▶ **Lemma 16.** *It holds that $c(OPT') \leq (1+\epsilon)c(OPT)$.*

3.3 Dynamic program

We present now a dynamic program that computes a solution ALG which satisfies that $c(ALG) \leq c(OPT') \leq (1+\epsilon)c(OPT)$. It has two stages: in the first stage, it intuitively

guesses the schedule for each $t \in I'_{\text{low}}$ and the maximally large intervals of I'_{high}. Note that for each $t \in I'_{\text{low}}$ we have that $W_{OPT}(t) \leq 2^{2k^2}m^2/\epsilon$ and hence the number of different configurations at time t can be bounded by some function $f(k, \epsilon, m)$. In the second stage, it computes a solution for each maximally large subinterval of I'_{high} that obeys the pack-property. To this end, note that if at a time τ during such a subinterval $[\tau_L, \tau_R)$ no job class is waiting then the number of released by infinished jobs at time τ can be bounded by some function $g(k, \epsilon, m)$. Thus, we can guess the schedule for the interval $[\tau, \tau + 1)$ in FPT-time and continue with the time point $\tau + 1$. Otherwise, the pack-property dictates which jobs we need to schedule next on the machines, i.e., we need to schedule a pack of jobs from the waiting job class of highest density (among all waiting job classes). We embed the guessing steps into a dynamic program that intuitively scans the time axis from left to right.

We assume w.l.o.g. that $\min_j r_j = 1$, $\max_j r_j = O(n^2k^2)$, and that OPT' finishes its last job before time $T := \max_j r_j + nk$, otherwise we can split the instance into independent subinstances. Similarly to the preemptive setting, we say that a *configuration for a time t* specifies which jobs are pending at time t, which jobs are being processed on some machine at time t, and the remaining processing time for each job of the latter kind at time t. Formally, a configuration is a vector $(\bar{p}_j)_{j \in J}$ such that $0 \leq \bar{p}_j \leq p_j$ and $\bar{p}_j \in \mathbb{N}$ for each $j \in J$ and an assignment of the currently running jobs to the machines, given by a function $\text{run} : \{1, ..., m\} \to J \cup \{\perp\}$ where $\text{run}(i) = \perp$ if at time t no job is running on machine i that was started before t. We allow only configurations in which the jobs in $\text{run}(\{1, ..., m\})$ are exactly the partially processed jobs, i.e, the jobs j with $0 < \bar{p}_j < p_j$. For each time t there is a configuration that corresponds to the status of OPT at time t. The following lemma is an analog of Lemma 5 of the non-preemptive case.

▶ **Lemma 17.** *For each time $t \in \mathbb{N}$ we can compute in time $(mk/\epsilon)^{O(mk^5)}n^{O(1)}$ a set $\mathcal{C}'(t)$ of at most $(mk/\epsilon)^{O(mk^5)}$ configurations such that if $t \in I'_{\text{low}}$ then one of them corresponds to OPT at time t. The set $\mathcal{C}'(T)$ contains only the configuration c_T in which all jobs have completed.*

Our first stage DP has a DP-cell for each tuple (t, c) where $t \in \{0, ..., T\}$ is a time and c is a configuration in $\mathcal{C}'(t)$ according to Lemma 17. A cell (t, c) corresponds to the subproblem of computing a schedule for the interval $[t, T)$ that schedules all jobs j that are pending at time t according to c or that satisfy $r_j > t$. Moreover, if a job j is running on a machine i at time t according to c, then its remaining part has to finish on machine i before another job can be processed on the same machine.

For the base case, the cell (T, c_T) stores an empty schedule. Suppose we are given a cell (t, c) with $t < T$. The reader may imagine that $t \in I'_{\text{low}}$ and that at time t the schedule OPT' is in configuration c. Intuitively, we guess the smallest time $t' > t$ with $t' \in I'_{\text{low}}$ and the configuration c' of OPT' at time t'. Formally, for each time $t' > t$ with $t' \in \mathbb{N}$, and each configuration $c' \in \mathcal{C}'(t')$ we do the following: in case that c and c' are inconsistent we discard the guess. Formally, note that the information about the partially processed jobs according to c and c' imply when those jobs have to be executed during $[t, t')$. We discard the guess if this would imply that a machine has to work on two jobs at the same time, if a job has to be started at different times or on different machines according to c and c', if in c' a job is pending that is already finished according to c, or if the resulting schedule would contradict Lemma 7. In case that $t' = t + 1$ we have that c and c' imply the schedule during $[t, t + 1)$, up to permutation of machines that work on unit size jobs during $[t, t + 1)$. We append to this schedule the schedule stored in the cell (t', c') for the time interval $[t', \infty)$.

If $t' > t+1$ then we create a subproblem for the second stage DP. This subproblem consists of

- the time interval $[t, t') =: [\tau_L, \tau_R)$,
- the set of jobs J' that have to be completely processed during $[\tau_L, \tau_R)$ according to c, and c', i.e., the jobs j that are ready at time τ_L according to c or satisfy $r_j > \tau_L$ and that are finished at time τ_R according to c', and
- the at most $2m$ jobs that are partially but not completely processed at time τ_L and τ_R according to c and c', together with their starting times and the information on which machine they are assigned. Note that since τ_L and τ_R are fixed, there are only $2k$ possibilities for the starting time of each such job. We denote by S the resulting schedule for these jobs.

Denote by (τ_L, τ_R, J', S) the resulting tuple which forms the input to our second stage DP. In this DP, intuitively we scan the time axis from left to right in order to compute a schedule for J' that satisfies the pack-property and that does not conflict with S. If at a time $\tau \in [\tau_L, \tau_R)$ no job class is waiting then we guess the schedule for the interval $[\tau, \tau+1)$ and continue with the time point $\tau+1$. Otherwise, we select a waiting job class $J_{\ell,\ell'}$ of highest density among all waiting job classes and schedule a pack of $J_{\ell,\ell'}$ as soon as possible.

Formally, our second stage DP has an entry (τ, c) for each combination of a time $\tau \in \{\tau_L, ..., \tau_R - 1\}$ and a configuration $c \in \mathcal{C}''(\tau)$ where $\mathcal{C}''(\tau)$ is the set of all configurations of the second stage DP for time τ, consisting only of jobs in J', such that for each job class $J_{\ell,\ell'}$ the total processing time of the ready jobs is less than $4m^2kk!$, i.e., there is no job class that is waiting at time τ. The subproblem for (τ, c) is the problem of computing a schedule S' for the interval $[\tau, \tau_R)$ for the jobs $j \in J'$ that are pending at time τ according to c or which satisfy $r_j > \tau$ and we require that this schedule is consistent with S, i.e., at each time τ no machine schedules one job according to S and another job according to S'. The cost of this schedule is the cost of all jobs in J' that finish during $[\tau, \tau_R)$.

▶ **Lemma 18.** *The number of DP-cells of the second stage DP is bounded by* $(mk)^{O(mk^4)}n^2k^2$.

Suppose we want to solve a subproblem (τ_L, τ_R, J', S) using our second stage DP. For the base case, let $\bar{c}_{\tau_R}$ denote the configuration in which $p'_j = 0$ for each job $j \in J'$. We define $(\tau_R, \bar{c}_{t_R})$ to be the empty schedule, for all other configurations $c'' \in \mathcal{C}''(\tau_R)$ we mark the DP-cell (τ_R, c'') as invalid. Intuitively, a cell (τ, c'') is marked invalid if there is no non-preemptive schedule for $[\tau, \tau_R)$ that finishes before time τ_R all jobs that are have to be processed during this interval according to c''. We describe now how to compute the entry for a cell (τ, c) such that $c \in \mathcal{C}''(\tau)$ and $\tau < \tau_R$. We guess the schedule for the time interval $[\tau, \tau+1)$, i.e., for each machine i that at time τ is not processing a previously started job we try all possibilities to choose a job to start and also allow for the possiblity to keep the machine idle during $[\tau, \tau+1)$. For jobs that start at time τ we require that within their job class they are the pending jobs with earliest release dates (like in OPT', see Lemma 7). Hence, there are only $\tilde{g}(k, \epsilon, m)$ possibilities for some function $\tilde{g}$.

For a given guess, let $\tilde{c}$ denote the resulting configuration at time $\tau+1$. If $\tilde{c} \in \mathcal{C}''(\tau+1)$ then we append the schedule stored in the cell $(\tau+1, \tilde{c})$, unless $(\tau+1, \tilde{c})$ was marked invalid in which case we discard our guess. If $\tilde{c} \notin \mathcal{C}''(\tau+1)$ then according to c' there must be a job class that is waiting. Let τ' be the smallest time step with $\tau+1 \leq \tau'$ such that a machine finishes a job on which it was working at time τ. In this case, let $J_{\ell,\ell'}$ be the job class with highest density that is waiting at time τ', breaking ties like in the construction of OPT'. Let $Q_{\ell,\ell'}$ denote the $mk!/\ell$ ready jobs with earliest release dates among the ready jobs in $J_{\ell,\ell'}$. For each machine i denote by $\tau(i)$ the time when it completes its job that it

executes during $[\tau, \tau+1)$ (according our guess and the jobs that started before τ) and let $\tau(i) := \tau + 1$ if there is no such job. On each machine i we schedule $k!/\ell$ jobs from $Q_{\ell,\ell'}$ during $[\tau(i), \tau(i)+k!)$. Let $\tilde{c}'$ denote the resulting configuration of the pending jobs in J' at time $\tau + 1 + k!$. If $\tilde{c}' \in \mathcal{C}''(\tau + 1 + k!)$ then the solution for our guess of the schedule during $[\tau, \tau+1)$ consists of the schedule for the jobs running at time τ according c, the just computed schedule for the jobs in $Q_{\ell,\ell'}$, and the solution stored in the cell $(\tau + 1 + k!, \tilde{c}')$, unless $(\tau + 1 + k!, \tilde{c}')$ was marked invalid in which case we discard our guess. Otherwise, there must be a waiting job class. In this case we repeat the process (i.e., identify a waiting jobs class and schedule a pack of its jobs as soon as possible) until we arrive at a time τ'' and a configuration $\tilde{c}'' \in \mathcal{C}''(\tau'')$ in which case we append the schedule in the DP-cell $(\tau'', \tilde{c}'')$ to the just computed schedule, or, again discard our guess in case that $(\tau'', \tilde{c}'')$ was marked invalid. In case that the resulting schedule contradicts S (i.e., according to the computed schedule a machine i has to work on a job $j \in J'$ at the same time as it has to work on some job $j' \notin J'$ according to S) we discard our initial guess for the interval $[\tau, \tau+1)$. Finally, in the cell (τ, c) we store the computed schedule of minimum cost over all guesses of the schedule for the interval $[\tau, \tau+1)$, or, in case that we discarded all guesses, we mark the cell (τ, c) as invalid.

Finally, let $\bar{c}_0$ denote the configuration in which $p'_j = p_j$ for each job $j \in J'$, i.e., the configuration of the jobs J' at time τ_L. We observe that $\bar{c}_0 \in \mathcal{C}''(\tau_L)$ since $c \in \mathcal{C}'(\tau_L)$ (where c is the configuration of the cell (t, c) of the first stage DP above) and hence the solution to the subproblem (τ_L, τ_R, J', S) is stored in the cell $(\tau_L, \bar{c}_0)$.

▶ **Theorem 19.** *For the problem of minimizing the weighted flow time on m identical machines with job release dates and without preemption, $P|r_j|\sum w_j F_j$, there is a $(1+\epsilon)$-approximation algorithm with a running time of $(mk/\epsilon)^{O(mk^5)} \cdot n^{O(1)}$, assuming that $p_j, w_j \in \{1, ..., k\}$ for each job j.*

References

1. Foto N. Afrati, Evripidis Bampis, Chandra Chekuri, David R. Karger, Claire Kenyon, Sanjeev Khanna, Ioannis Milis, Maurice Queyranne, Martin Skutella, Clifford Stein, and Maxim Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, pages 32–44. IEEE, 1999.
2. Yossi Azar and Noam Touitou. Improved online algorithm for weighted flow time. *CoRR*, abs/1712.10273, 2017. `arXiv:1712.10273`.
3. Nikhil Bansal. Minimizing flow time on a constant number of machines with preemption. *Oper. Res. Lett.*, 33(3):267–273, 2005. `doi:10.1016/j.orl.2004.07.008`.
4. Nikhil Bansal and Kedar Dhamdhere. Minimizing weighted flow time. *ACM Trans. Algorithms*, 3(4), 2007. `doi:10.1145/1290672.1290676`.
5. Nikhil Bansal and Kirk Pruhs. The geometry of scheduling. *SIAM Journal on Computing*, 43(5):1684–1698, 2014.
6. Jatin Batra, Naveen Garg, and Amit Kumar. Constant factor approximation algorithm for weighted flow time on a single machine in pseudo-polynomial time. *CoRR*, abs/1802.07439, 2018. `arXiv:1802.07439`.
7. Chandra Chekuri and Sanjeev Khanna. A ptas for minimizing weighted completion time on uniformly related machines. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *Automata, Languages and Programming*, pages 848–861, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

8 Chandra Chekuri and Sanjeev Khanna. Approximation schemes for preemptive weighted flow time. In *Proceedings of the thiry-fourth annual ACM Symposium on Theory of computing*, pages 297–305. ACM, 2002.

9 Chandra Chekuri, Sanjeev Khanna, and An Zhu. Algorithms for minimizing weighted flow time. In *Proceedings of the thirty-third annual ACM Symposium on Theory of computing*, pages 84–93. ACM, 2001.

10 Leah Epstein and Asaf Levin. Minimum total weighted completion time: Faster approximation schemes. *CoRR*, abs/1404.1059, 2014. `arXiv:1404.1059`.

11 Naveen Garg, Amit Kumar, and V. N. Muralidhara. Minimizing total flow-time: The unrelated case. In *Proceedings of the 19th International Symposium on Algorithms and Computation (ISAAC 2008)*, pages 424–435. Springer, 2008. `doi:10.1007/978-3-540-92182-0_39`.

12 Fabrizio Grandoni, Stefan Kratsch, and Andreas Wiese. Parameterized approximation schemes for independent set of rectangles and geometric knapsack, 2017. Unpublished.

13 Hans Kellerer, Thomas Tautenhahn, and Gerhard Woeginger. Approximability and nonapproximability results for minimizing total flow time on a single machine. *SIAM Journal on Computing*, 28(4):1155–1166, 1999.

14 Dusan Knop and Martin Koutecký. Scheduling meets n-fold integer programming. *CoRR*, abs/1603.02611, 2016. `arXiv:1603.02611`.

15 Stefano Leonardi and Danny Raz. Approximating total flow time on parallel machines. In *Proceedings of the twenty-ninth annual ACM Symposium on Theory of Computing*, pages 110–119. ACM, 1997.

16 Joseph Y-T. Leung. *Handbook of scheduling: algorithms, models, and performance analysis*. CRC Press, 2004.

17 Dániel Marx. Parameterized complexity and approximation algorithms. *The Computer Journal*, 51(1):60–78, 2008.

18 Matthias Mnich and Andreas Wiese. Scheduling and fixed-parameter tractability. *Math. Program.*, 154(1-2):533–562, 2015.

19 Andreas Wiese. A (1+epsilon)-approximation for unsplittable flow on a path in fixed-parameter running time. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, pages 67:1–67:13, 2017.

A Omitted proofs

A.1 Proof of Lemma 4

We define a partially fractional objective function: for a given solution S and a value $t \in \Gamma$ let $\tilde{W}_S(t) = \sum_{j \in J: r_j \leq t} \frac{w_j}{p_j} p_j^S(t)$ where $p_j^S(t)$ denotes the total length of each job j that has not been finished by time t in S. We define a new objective function by $\sum_{t \in I_{\text{low}} \cap \Gamma} \epsilon W(t) + \sum_{t \in I_{\text{high}} \cap \Gamma} \epsilon \tilde{W}(t)$. We have that $\sum_{t \in I_{\text{low}} \cap \Gamma} \epsilon W_{OPT'}(t) + \sum_{t \in I_{\text{high}} \cap \Gamma} \epsilon \tilde{W}_{OPT'}(t) \leq \sum_{t \in I_{\text{low}} \cap \Gamma} \epsilon W_{OPT}(t) + \sum_{t \in I_{\text{high}} \cap \Gamma} \epsilon \tilde{W}_{OPT}(t)$ since the solution values of our fractional schedules in the maximally large intervals of I_{high} are a lower bound on the respective cost of OPT in these intervals.

On the other hand, for each $t \in I_{\text{high}} \cap \Gamma$ we have that $W_{OPT'}(t) \leq \tilde{W}_{OPT'}(t) + 3mk^3 \leq \tilde{W}_{OPT'}(t) + \epsilon W_{OPT}(t)$ since $W_{OPT}(t) > 3mk^3/\epsilon^2$ and for each job class and each time t there are at most $3mk/\epsilon$ jobs that have been started in OPT' but not yet completed: at most mk/ϵ jobs j for which $0 < \bar{p}_j(t) < p'_j$ (where p'_j denotes the amount of work that OPT finishes on job j during the maximally large subinterval of I_{high} that contains t), at most mk/ϵ jobs j with $\bar{p}_j(t) = p'_j$ but $p'_j < p_j$ (those jobs are partially scheduled at time t_2 in OPT and by Lemma 2 there can be only mk/ϵ of those) and at most mk/ϵ jobs j with

$\bar{p}_j(t) = 0$ but that have already been partially processed in OPT at time t_1 (again, due to Lemma 2). This yields

$$\begin{aligned}\int_t W_{OPT'}(t)dt &= \sum_{t\in\Gamma} \epsilon W_{OPT'}(t)\\ &\leq \sum_{t\in\Gamma\cap I_{\text{low}}} \epsilon W_{OPT'}(t) + \sum_{t\in\Gamma\cap I_{\text{low}}} \left(\epsilon\tilde{W}_{OPT'}(t) + \epsilon^2\cdot W_{OPT}(t)\right)\\ &\leq \sum_{t\in\Gamma\cap I_{\text{low}}} \epsilon W_{OPT'}(t) + \sum_{t\in\Gamma\cap I_{\text{low}}} \left(\epsilon W_{OPT}(t) + \epsilon^2\cdot W_{OPT}(t)\right)\\ &\leq c(OPT) + \epsilon\cdot c(OPT)\\ &\leq (1+\epsilon)c(OPT).\end{aligned}$$

A.2 Proof of Lemma 9

We will transform OPT step by step into a schedule for J' satisfying the pack-property. We consider the timesteps $t \in \{t_1, ..., t_2 - 1\}$ in increasing order. Define $OPT_{t_1-1} := OPT$. For each timestep $t \in \{t_1, ..., t_2 - 1\}$ we take the schedule OPT_{t-1}, assuming inductively that $OPT_{t'}$ fulfills the pack-property for each $t' \in \{t_1, ..., t-1\}$. If in OPT_{t-1} at time t there is no waiting job class $J_{\ell,\ell'}$ then we define $OPT_t := OPT_{t-1}$ and hence OPT_t fulfills the pack-property at each $t' \in \{t_1, ..., t\}$. Otherwise, let $J_{\ell,\ell'}$ denote a waiting job class with maximum density ℓ'/ℓ, breaking ties in an arbitrary fixed order. Denote by $\tilde{J}_{\ell,\ell'}$ the at least $4m^2k^2k!$ jobs from $J_{\ell,\ell'}$ that are ready at time t. Hence, there must be a machine i^* such that jobs from $\tilde{J}_{\ell,\ell'}$ with a total processing time of $4mkk!$ are scheduled on i^* during $[t, t_2)$. Note that this implies in particular that $t_2 \geq t + 4m^2k^2k!$. For i let $t(i)$ be the earliest time $t' \geq t$ when machine i starts a new job or is idle. We change the order of the jobs on i^* such that after $t(i^*)$ we first schedule the jobs in $\tilde{J}_{\ell,\ell'}$ on i^* and then all other jobs that are scheduled on i^* in OPT_{t-1} in the same order as in OPT_{t-1}. Next, we swap some of the jobs in $\tilde{J}_{\ell,\ell'}$ on i^* to the other machines such that each machine i schedules $k!/\ell$ jobs from $\tilde{J}_{\ell,\ell'}$.

▶ **Claim 20.** *For each machine $i \neq i^*$ there is a set of jobs $J(i)$ that i starts and completes during $[t, t + 2k\cdot k! + k)$ such that there are $k!$ time units in $[t, t + 2k\cdot k!)$ during which machine i either works on a job in $J(i)$ or is idle.*

Proof. It suffices to define $J(i) := \emptyset$ if the total amount of idle time on machine i during $[t(i), t(i) + 2k\cdot k!)$ is at least $k!$. Otherwise, during $[t(i), t(i) + 2k\cdot k! + k)$ machine i starts and completes jobs with a total processing time of at least $kk!$ and hence there must be an integer $k' \leq k$ such that i starts and completes at least $k!/k'$ jobs of size exactly k'. We define $J(i)$ to be $k!/k'$ of these jobs. ◀

Since machine i^* schedules jobs from $\tilde{J}_{\ell,\ell'}$ with a total processing time of at least $4mkk! \geq mk! + 2kk! + k$, there must be $mk!/\ell$ of them which start at time $t + 2kk! + k$ or later. For each machine $i \neq i^*$ we swap the jobs $J(i)$ with $k!/\ell$ jobs on i^* from $\tilde{J}_{\ell,\ell'}$ that start at time $t + 2k\cdot k! + k$ or later. On machine i, after time $t(i)$ we first schedule the jobs that were previously on i^* and then schedule the remaining jobs in the same order as in OPT_{t-1}. No job is started before its release date since all jobs in $\tilde{J}_{\ell,\ell'}$ are ready at time t. Also, since in OPT_{t-1} there are $k!$ time units during $[t, t + 2k\cdot k! + k)$ in which i either works on a job in $J(i)$ or is idle, each job in J' on i completes by time $t_2(i)$ the latest. On i^* we first schedule all jobs from $\tilde{J}_{\ell,\ell'}$ and then the jobs from the sets $J(i)$ in an arbitrary order. Since each job in $J(i)$ starts before time $t + 2k\cdot k! + k$ in OPT_{t-1} it starts no earlier than its release date (after the swap).

A.3 Proof of Lemma 14

Based on OPT, we construct a schedule $\overline{OPT}$ for the jobs in J_{p} that fulfills that $W_{\overline{OPT}}(t) \leq W_{OPT}(t) + 4m^2k^5k!$ for each t and in which no job j in a pack Q_r is executed before time t'_r. Then the claim of the lemma follows since for each such schedule S and each t we have that $\tilde{W}_{FRAC}(t) \leq W_S(t)$. Consider a job class $J_{\ell,\ell'}$ and let $Q_1, Q_2, ...$ be its corresponding packs and let $t'_1, t'_2, ...$ be there corresponding times t'. To construct $\overline{OPT}$, for each pack Q_s we schedule the jobs in Q_s during the times when OPT schedules the jobs in $Q_{s+4m^2k^2}$. Observe that for each job $j \in Q_{s+4m^2k^2}$ it holds that $r_j \geq t'_s$ and hence after this swap each job in Q_s is started no earlier than t'_s. We remove the jobs in the last $4m^2k^2$ packs of $J_{\ell,\ell'}$ from the schedule, one may imagine that we schedule them at the very end after t_2. So intuitively, at each time t the schedule $\overline{OPT}$ is at most $4m^2k^2$ packs "behind" the schedule OPT. The total weight of the jobs in $4m^2k^2$ packs is bounded by $k \cdot 4m^2k^2k!$. This implies that $W^{J_{\ell,\ell'}}_{\overline{OPT}}(t) \leq W^{J_{\ell,\ell'}}_{OPT}(t) + 4m^2k^3k!$ for each t where $W^{J_{\ell,\ell'}}_{OPT}(t)$ and $W^{J_{\ell,\ell'}}_{\overline{OPT}}(t)$ denote the pending weight of the jobs in $J_{\ell,\ell'}$ at time t, respectively. Since there are at most k^2 jobs classes, this implies that $W_{\overline{OPT}}(t) \leq W_{OPT}(t) + 4m^2k^5k!$ and hence $\tilde{W}_{FRAC}(t) \leq W_{OPT}(t) + 4m^2k^5k!$.

A.4 Proof of Lemma 15

The second inequality follows from the first one by a geometric sum argument. We prove the first inequality. The claim is clearly true for $t = t_1$. For all other t we prove the lemma by induction over the groups $J^{(1)}_{\mathrm{p}}, J^{(2)}_{\mathrm{p}}, ..., J^{(\gamma)}_{\mathrm{p}}$. For the base case, consider the group $J^{(\gamma)}_{\mathrm{p}}$. The jobs in $J^{(\gamma)}_{\mathrm{p}}$ have the maximum density among all jobs in the instance. For this group, we prove the claim by induction over t. More precisely, we prove that $W^{J^{(\gamma)}_{\mathrm{p}}}_{OPT'_{\mathrm{p}}}(t) \leq \tilde{W}^{J^{(\gamma)}_{\mathrm{p}}}_{FRAC}(t) + m \cdot k!$ for each t. The claim is true for $t = t_1$. Consider an arbitrary time t and suppose that the claim is true for each time $t' < t$. If during $[t-1, t)$ in the schedule OPT'_{p} one machine is working on a job in $J^{(\gamma)}_{\mathrm{p}}$ then all machines are working on a job in $J^{(\gamma)}_{\mathrm{p}}$ and thus the claim is also true for time t. Similarly, if during $[t-1, t)$ the schedule $FRAC$ is not working on a job in $J^{(\gamma)}_{\mathrm{p}}$ then the claim is also true for time t. Otherwise, assume that during $[t-1, t)$ in the schedule OPT'_{p} no machine is working on a job in $J^{(\gamma)}_{\mathrm{p}}$ but in $FRAC$ some machine is working on a job in $J^{(\gamma)}_{\mathrm{p}}$. Hence, in $FRAC$ all machines are working on a job in $J^{(\gamma)}_{\mathrm{p}}$ during $[t-1, t)$. Thus, there is a pack Q_r consisting of jobs in $J^{(\gamma)}_{\mathrm{p}}$ with $t_r \leq t-1$ of density γ with $t_r \leq t-1$ but OPT'_{p} does not yet work on Q_r. This must be because OPT'_{p} has already started processing jobs from another pack $Q_{r'}$ but it has not yet finished them. Suppose that OPT'_{p} started the pack $Q_{r'}$ at a time $\hat{t} < t-1$. Then, however, $\hat{t} \geq t - 1 - (k! - 1) = t - k!$ and $W^{J^{(\gamma)}_{\mathrm{p}}}_{OPT'_{\mathrm{p}}}(\hat{t}) = 0$. The schedule $FRAC$ processes the jobs in $J^{(\gamma)}_{\mathrm{p}}$ at least as fast as OPT'_{p} and thus also $\tilde{W}^{J^{(\gamma)}_{\mathrm{p}}}_{FRAC}(\hat{t}) = 0$. The schedule $FRAC$ can schedule at most one job from $J^{(\gamma)}_{\mathrm{p}}$ at a time on each machine. Hence, $W^{J^{(\gamma)}_{\mathrm{p}}}_{OPT'_{\mathrm{p}}}(t) \leq \tilde{W}^{J^{(\gamma)}_{\mathrm{p}}}_{FRAC}(t) + mk \cdot k! = \tilde{W}^{J^{(\gamma)}_{\mathrm{p}}}_{FRAC}(t) + 2^{\gamma-r}mk \cdot k!$.

Now assume that there is a value g such that the claim is true for each $g' > g$. Consider the group $J^{(g)}_{\mathrm{p}}$. Again, we prove the claim for $J^{(g)}_{\mathrm{p}}$ by induction over t. At time $t = t_1$ the claim is clearly true. Suppose that there is a time t such that for each $t' < t$ the claim is true. The claim is immediately true for t if in OPT'_{p} at least one machine (and hence all machines) work on a job of group $J^{(g)}_{\mathrm{p}}$ during $[t-1, t)$ or if in $FRAC$ no machine is working on a job in $J^{(g)}_{\mathrm{p}}$ during $[t-1, t)$. Therefore, let us assume that the latter statements are both not true. Let $t'' \leq t$ be the latest point in time before t when $W^{J^{(g)}_{\mathrm{p}}}_{OPT'_{\mathrm{p}}}(t'') \leq \tilde{W}^{J^{(g)}_{\mathrm{p}}}_{FRAC}(t'')$. If $t'' = t$ there is nothing to show so assume that $t'' < t$. Hence, during $[t'', t''+1)$ in the

schedule $FRAC$ all machines work on a job in $J_{\mathrm{p}}^{(g)}$ but in OPT'_{p} no machine works on a job in $J_{\mathrm{p}}^{(g)}$. In particular, at time t'' there is a pack Q_r with jobs in $J_{\mathrm{p}}^{(g)}$ with $t'_r \leq t''$ but at time t'' it has neither been completely processed by OPT'_{p} nor by $FRAC$. Also, OPT'_{p} does not process jobs from Q_r during $[t'', t''+1)$ (otherwise this would contradict our definition of t'') and hence Q_r has not been started in OPT'_{p}. Since OPT'_{p} and $FRAC$ order the packs from each group $J_{\mathrm{p}}^{(g')}$ according to the artificial release dates t'_r this implies that also in $FRAC$ subpack Q_r has not been started yet.

Also, at time t'' in $FRAC$ there is no job available from a group $J_{\mathrm{p}}^{(g')}$ with $g' > g$ and thus $\tilde{W}_{FRAC}^{J_{\mathrm{p}}^{(g')}}(t'') = 0$ for each such g'. Hence, the induction hypothesis implies that $\tilde{W}_{OPT'_{\mathrm{p}}}^{J_{\mathrm{p}}^{(g')}}(t'') < 2^{\gamma-g'} m \cdot k!$ for each such g'. Let $\bar{J}$ denote all jobs in packs $Q_{r'}$ with $t'_{r'} \in [t'', t)$ and which are contained in a group $J_{\mathrm{p}}^{(g')}$ with $g' > g$. Since $FRAC$ works on a jobs in $J_{\mathrm{p}}^{(g)}$ during $[t-1, t)$ all jobs in $\bar{J}$ finish in $FRAC$ during $[t'', t)$. Hence, during $[t'', t)$ in the schedule $FRAC$ there are at most $t - t'' - p(\bar{J})/m$ units of time during which at least one (and hence all) machines work on jobs in a group $J_{\mathrm{p}}^{(g')}$ with $g' > g$. At each time $\hat{t} \in [t'', t)$ we have that $W_{OPT'_{\mathrm{p}}}^{J_{\mathrm{p}}^{(g)}}(\hat{t}) \geq 1$ since otherwise this would contradict our choice of t''. Hence, if at a time $\hat{t} \in [t'', t)$ the schedule OPT'_{p} does not work on a job in $J_{\mathrm{p}}^{(g)}$ then OPT'_{p} works on a job in a pack that is partially processed at time t'' or on a job from a group $J_{\mathrm{p}}^{(g')}$ with $g' > g$. Recall that $\tilde{W}_{OPT'_{\mathrm{p}}}^{J_{\mathrm{p}}^{(g')}}(t'') < 2^{\gamma-g'} m \cdot k!$ for each $g' > g$. Hence, the total time, summed up over all machines, during $[t'', t)$ when OPT'_{p} does not work on a job in $J_{\mathrm{p}}^{(g)}$ is bounded by $m \cdot k! + p(\bar{J}) + \sum_{g':g'>g} 2^{\gamma-g'} m \cdot k!$. Therefore, OPT'_{p} finishes at least $m(t-t'') - p(\bar{J}) - m \cdot k! - \sum_{g':g'>g} 2^{\gamma-g'} m \cdot k!$ units of work of jobs in $J_{\mathrm{p}}^{(g)}$ during $[t'', t)$. Since $W_{OPT'_{\mathrm{p}}}^{J_{\mathrm{p}}^{(g)}}(t'') \leq \tilde{W}_{FRAC}^{J_{\mathrm{p}}^{(g)}}(t'')$ we have that $W_{OPT'_{\mathrm{p}}}^{J_{\mathrm{p}}^{(g)}}(t) - \tilde{W}_{FRAC}^{J_{\mathrm{p}}^{(g)}}(t) \leq m(t-t'') - p(\bar{J}) - (m(t-t'') - p(\bar{J}) - m \cdot k! - \sum_{g':g'>g} 2^{\gamma-g'} m \cdot k!) = 2^{\gamma-g} m \cdot k!$ and therefore $W_{OPT'_{\mathrm{p}}}^{J_{\mathrm{p}}^{(g)}}(t) \leq \tilde{W}_{FRAC}^{J_{\mathrm{p}}^{(g)}}(t) + 2^{\gamma-g} m \cdot k!$.

A.5 Proof of Lemma 16

We first prove that for each time $t \in I'_{\mathrm{high}}$ it holds that $W_{OPT'}(t) \leq (1+\epsilon) W_{OPT}(t)$. Let $t \in I'_{\mathrm{high}}$. We have that

$$
\begin{aligned}
W_{OPT'}(t) &= W^{\mathrm{np}}_{OPT'}(t) + W^{\mathrm{p}}_{OPT'}(t) \\
&\overset{\text{Proposition 10}}{\leq} 4m^2k^4k! + W^{\mathrm{p}}_{OPT'}(t) \\
&\overset{\text{Lemma 12}}{\leq} 4m^2k^4k! + W_{OPT''_{\mathrm{p}}}(t) + k^2 m \\
&\leq 5m^2k^4k! + \sum_{g=1}^{\gamma} W_{OPT''_{\mathrm{p}}}^{J_{\mathrm{p}}^{(g)}}(t) \\
&\overset{\text{Lemma 15}}{\leq} 5m^2k^4k! + \sum_{g=1}^{\gamma} \left(\tilde{W}_{FRAC}^{J_{\mathrm{p}}^{(g)}}(t) + 2^{\gamma-g} m \cdot k! \right) \\
&\leq 5m^2k^4k! + 2^{\gamma} m \cdot k! + \tilde{W}_{FRAC}(t) \\
&\overset{\text{Lemma 14}}{\leq} 5m^2k^4k! + 2^{k^2} m^2 \cdot k! + W_{OPT}(t) + 4m^2k^5k! \\
&\leq \epsilon W_{OPT}(t) + W_{OPT}(t) \\
&= (1+\epsilon) W_{OPT}(t).
\end{aligned}
$$

Since $W_{OPT'}(t) = W_{OPT}(t)$ for each $t \in I'_{\text{low}}$ we conclude that $c(OPT') = \sum_t W_{OPT'}(t) \leq (1+\epsilon)\sum_t W_{OPT}(t) = (1+\epsilon)c(OPT)$.

A.6 Proof of Lemma 17

Since $W_{OPT}(t) \leq 2^{2k^2}m^2/\epsilon$ for each $t \in I'_{\text{low}}$ there can be at most $2^{2k^2}m^2/\epsilon$ pending jobs at time t. For each of them, there are k^2 options to which class it belongs to and k options for its remaining processing time. For each job class, there are $m!$ possibilities for the identities of the at most m partially scheduled jobs and additionally $m!$ possibilities for the machines on which the partially processed jobs run. This yields $\left(1 + 2^{2k^2}m^2/\epsilon\right)^{k^3} (m!)^{k^2+1} = (mk/\epsilon)^{O(mk^5)}$ possible configurations for the pending jobs, which of them are running at time t on the machines, and how much of them has already been processed.

A.7 Proof of Lemma 18

For each job class there can be at most $4m^2kk!$ pending jobs and there are k options for their remaining processing time. This yields $(1 + 4m^2kk!)^k$ options per job class. There are k^2 job classes and there are $m!$ possibilities to distribute the partially processed jobs on the m machines. This yields $m!(1 + 4m^2kk!)^{k^3} = (mk)^{O(mk^4)}$ options in total for each $\tau \in \{\tau_L, ..., \tau_R\}$.

A.8 Proof of Theorem 19

We first prove the correctness of the second stage DP and then based on this the correctness of the first stage DP. Given an instance (τ_L, τ_R, J', S) of the second stage DP such that $\tau_L, \tau_R \in I'_{\text{low}}$, S is the schedule of all jobs that are partially processed at times τ_L or τ_R in OPT', and J' is the set of jobs that OPT' completely finishes during $[\tau_L, \tau_R)$. We prove that the second stage DP computes a schedule S' for the interval $[\tau_L, \tau_R)$ in which

- all jobs in J' are completed
- at each time $\tau \in [\tau_L, \tau_R)$ each machine i either works on a job $j \in J'$, or works on some job $j \notin J'$ according to S, or works on no job at all, and
- the cost of the jobs J' in S' is bounded by the cost of the jobs in J' in OPT', i.e., $\sum_{j\in J'} w_j(C_j^{S'} - r_j) \leq \sum_{j\in J'} w_j(C_j^{OPT'} - r_j)$ where $C_j^{S'}$ and $C_j^{OPT'}$ denote the completion times of job j in S' and OPT', respectively.

In this proof, whenever we refer to a configuration c of OPT' at a time τ, we mean the configuration of OPT' restricted to the jobs in J'. We prove by induction for each cell (τ, c) of the second stage DP such that OPT' is in configuration c at time τ and $c \in \mathcal{C}''(\tau)$ that the cost of the schedule for the interval $[\tau, \tau_R)$ stored in (τ, c) is upper-bounded by the cost of OPT' during $[\tau, \tau_R)$. The claim is true by definition for the base case where $\tau = \tau_R$ since then both schedules are empty. Suppose that there is a value $\tau \in [\tau_L, \tau_R)$ such that the claim is true for each (τ', c') with $\tau' \in [\tau_L, \tau_R)$ and $\tau' > \tau$ such that $c' \in \mathcal{C}''(\tau')$ and OPT' is in configuration c' at time τ'. We show that the claim is true for the cell (τ, c) such that OPT' is in configuration c at time τ in case that $c \in \mathcal{C}''(\tau)$. Let c be the configuration of OPT' at time τ. In case that $c \notin \mathcal{C}''(\tau)$ there is nothing to show. Assume that $c \in \mathcal{C}''(\tau)$. Among all our guesses for the schedule of the interval $[\tau, \tau+1)$ we enumerate one guess that coincides with the schedule of OPT' during $[\tau, \tau+1)$. Let c' denote the resulting configuration at time $\tau+1$. If $c' \in \mathcal{C}''(\tau+1)$ then by induction the schedule corresponding to this guess (i.e., the guessed schedule for $[\tau, \tau+1)$ plus the schedule for $[\tau+1, \tau_R)$ stored in the cell $(\tau+1, c')$) has a cost of at most the cost of OPT' during $[\tau, \tau_R)$, i.e., the cost of the jobs finishing during

$[\tau, \tau_R)$ in OPT'. If $c' \notin \mathcal{C}''(\tau+1)$ then we schedule a pack or several packs of jobs until we reach a time τ'' and a configuration $\tilde{c}'' \in \mathcal{C}''(\tau'')$. Since OPT' satisfies the pack property, we produce a schedule satisfying the pack property, and at each time $\hat{\tau} \in \{\tau+1, ..., \tau''-1\}$ there is a waiting job class or all machines are busy with pack jobs, the resulting schedule for the interval $[\tau, \tau'')$ is identical to the schedule of OPT' during $[\tau, \tau'')$. Hence, by induction we know that the resulting schedule for the interval $[\tau, \tau_R)$ for this guess has a cost that is at most the cost of OPT' during the interval $[\tau, \tau_R)$. This completes the proof for the second stage DP.

For the first stage DP, one can prove correctness with a similar induction as in the proof of Theorem 6, using that for a cell (t, c) with $t+1 \notin I'_{\text{low}}$ the DP guesses the earliest time t' with $t' > t$ and $t' \in I'_{\text{low}}$, the configuration c' of OPT' at time t', and that then the second stage DP computes a schedule for the interval $[t, t')$ which completes the same set of jobs that OPT' completes during $[t, t')$ and whose cost is upper-bounded by the cost of OPT' for these jobs.

It remains to prove the running time of the algorithm. The number of DP cells of the first stage DP is bounded by $(mk/\epsilon)^{O(mk^5)}O(n^2k^2)$, using Lemma 17 and that $T \leq O(n^2k^2)$. In order to compute the entry of a cell (t, c) of the first stage DP, we enumerate over $(mk/\epsilon)^{O(mk^5)}n^2k^2$ possibilities for (t', c'). In case that $t' = t+1$ we can compute the schedule for $[t, t+1)$ in time $n^{O(1)}$.

If $t' > t+1$ we additionally solve an instance of the second stage DP. The number of DP-cells of the second stage DP is bounded by $(mk)^{O(mk^4)}n^2k^2$. To compute the entry of one cell (t, c) we enumerate over $k^{O(m)}$ schedules for the interval $[t, t+1)$ and then possibly compute a schedule for pack jobs in time $n^{O(1)}$. Hence, we solve the resulting instance of the second stage DP in time $(mk)^{O(mk^4)}n^2k^2k^{O(m)}n^{O(1)} = (mk)^{O(mk^4)}n^{O(1)}$. Therefore, the schedule of a DP cell of the first stage DP can be computed in time $(mk/\epsilon)^{O(mk^5)}n^2k^2 \cdot (mk)^{O(mk^4)}n^{O(1)} = (mk/\epsilon)^{O(mk^5)}n^{O(1)}$ and thus the entries of all cells can be computed in time $(mk/\epsilon)^{O(mk^5)}n^{O(1)}(mk/\epsilon)^{O(mk^5)}n^{O(1)} = (mk/\epsilon)^{O(mk^5)}n^{O(1)}$.